Proposta de Modelo para o Desenvolvimento de Interface de Controle Manual em Ambiente de Simulação Utilizando os Softwares *Player* e *Stage*

Thiago da Silva Almeida, Nathanael Oliveira Vasconcelos,
Fabricio de Oliveira Fonseca e Alcides Xavier Benicasa
Departamento de Sistemas de Informação - DSI
Universidade Federal de Sergipe - UFS
Itabaiana-SE, Brasil
e-mail: thiago7a@gmail.com; nathan-vasconcelos@hotmail.com;
tr3mere@hotmail.com; alcides@ufs.br

***Resumo*—Este trabalho tem como principal objetivo apresentar todas as etapas necessárias para o controle da navegação de um robô inserido em um mundo criado a partir de um ambiente de simulação. Para isso foi necessário realizar estudos teóricos relacionados à pesquisa, importância e funcionamento do ambiente de simulação escolhido. Sendo finalizado com desenvolvimento do software cuja navegação é controlada manualmente através de botões desenvolvidos em uma interface gráfica.**

***Keywords*-Ambiente de simulação, controle de navegação, robótica, *Player* e *Stage*;**

I. Introdução

O estudo da robótica móvel é um tema bastante relevante e atual, pois se torna cada vez mais importante ter uma máquina capaz de chegar a lugares onde o ser humano não chega, e que suas ações sejam semelhantes às de um ser humano. Para criar uma maquina de estados autômatos, precisamos conhecer os equipamentos necessários, ambientes ideais para desenvolver um estudo que seja suficiente para fomentar pesquisas posteriores e diminuir o custo de desenvolvimento de projetos.

Objetivo desta pesquisa é, então, apresentar a importância do uso de ferramentas de simulação e o funcionamento específico da ferramenta de simulação utilizada neste estudo, mostrando a importância da robótica na sociedade e na computação, o funcionamento de um robô de uma maneira geral e concluindo com o resultado do software desenvolvido.

II. Ambiente de Simulação *Player* e *Stage*

O estudo de robótica móvel é um tema atual e que possui grande importância, é perceptível o crescimento de estudos, pesquisas nessa área nas últimas décadas. A cada dia surgem aplicações práticas de robôs móveis nas mais diferentes atividades em nossa sociedade, como por exemplo, na área doméstica, industrial, urbana, militar, transporte, segurança, demonstrando assim, sua importância, os interesses econômicos envolvidos e a necessidade do constante estudo.

Construir um robô ou comprá-lo exige um custo que a pesquisa ou pesquisador pode não ter condições de arcar primordialmente, portanto, o ambiente de simulação é um passo necessário e viável para compreensão das regras e do uso do material disponível no ambiente real.

De acordo com WOLF and Trindade Jr. (2009), o uso de um ambiente virtual de simulação de robôs é uma ferramenta muito viável e poderosa, e podem ser destacadas as seguintes características:

- Economia de tempo, pois podemos realizar um maior número de experimentos através de simulação, nos quais a realização de um experimento requer um menor tempo para configurar o experimento (não há necessidade de recarregar baterias e posicionar equipamentos e objetos), além de se poder "acelerar o tempo" do relógio virtual na execução das simulações;
- Evitar danos ao robô, pois através das simulações pode-se verificar previamente as situações que podem provocar danos ao robô, devido, por exemplo, a fortes colisões, acionamento indevido dos motores, ou exposição do robô à ambientes perigosos para testes;
- Evitar acidentes e aumentar a segurança, pois através da simulação pode-se realizar diversos testes visando garantir uma maior segurança e robustez do sistema robótico, evitando assim a incidência de acidentes com pessoas e com elementos presentes no ambiente de atuação do robô permitindo, inclusive, uma melhor análise de como adicionar novos dispositivos de salvaguarda em hardware e software que permitam aumentar a segurança e confiabilidade dos robôs;
- Aperfeiçoamento do hardware e software, uma vez que se pode, através de simulações, testar diferentes configurações de hardware, bem como testar e avaliar novas implementações e ajustes nos parâmetros do software de controle, permitindo assim uma melhoria do sistema como um todo e a otimização do uso dos recursos disponíveis a fim de obter uma maior eficiência do sistema robótico;
- Economia de recursos financeiros, pois diversos testes podem ser realizados antes de ser implementado fisica-

mente no robô.

Foi escolhida a ferramenta e plataforma que se ”tornou-se padrão na comunidade de robotica em codigo aberto”, (COLLETT and GERKEY, 2005). O *Player e Stage*, conhecidos como *Player/Stage*, juntos fazem a simulação do robô em um ambiente virtualizado, tornando as pesquisas mais rápidas e com menor custo.

Atualmente o desenvolvimento do *Player* conta com a colaboração de diversos pesquisadores das mais diversas instituições. Por ser um sistema de código aberto e de livre distribuição (compatível com o Linux), é um aplicativo para controle de robôs móveis amplamente utilizado por universidades, institutos de pesquisa e empresas em diversos países. Seu desenvolvimento foi iniciado em 2000 por pesquisadores da University of Southern Califórnia - EUA (WOLF and Trindade Jr., 2009).

A. Camada de Abstração de Hardware

De acordo com OWEN (2010), o *Player/Stage* funciona como uma camada de abstração de hardware. Isso significa que ele comunica-se com os *bits* de hardware do robô (como uma garra ou uma câmera, por exemplo) e permite ao usuário controlá-los com seu código, ou seja, não precisa se preocupar como as várias partes do trabalho do robô. O *Stage* é um plugin para o *Player* que identifica o que ele está informando para executar, e retorna novas instruções para a simulação do robô. Ele também simula os dados do sensor e envia para o *Player*, o qual por sua vez faz com que os dados dos sensores disponíveis retornem para o código.

Múltiplos robôs e sensores podem ser simulados simultaneamente, controlados por um ou mais clientes. Devido a sua alta eficiência, o *Stage* é capaz de simular dezenas de robôs operando simultaneamente em um único computador. Os robôs simulados pelo *Stage* são baseados no Pioneer (robô fabricado pela empresa *MobileRobots*). Essa plataforma é amplamente utilizada em laboratórios de pesquisa e ensino. O *Stage* simula o deslocamento dos robôs e também os sensores, como odômetros, lasers, sonares, câmeras e garras (WOLF and Trindade Jr., 2009). Podemos concluir o processo de simulação por três partes:

- O código que se comunica com *Player*;
- O *Player* recebe o código e envia as instruções para um robô e o robô recebe os dados do sensor e retorna para o código;
- O *Stage* recebe instruções do *Player* e move o robô simulado em um mundo simulado, que recebe os dados do sensor do robô na simulação e envia este *Player*.

A Figura 1 mostra claramente como funciona o processo de simulação explicado anteriormente.

B. Estrutura de Arquivos do Ambiente de Simulação

O estudo sobre ambiente de simulação foi baseado no trabalho de RIBEIRO et al. (2001). Os autores descrevem que para criação do ambiente de simulação são necessários

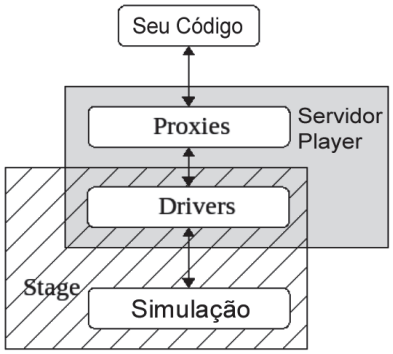

Figura 1. Estrutura do processo de simulação do Player/Stage.

três tipos de arquivo, o arquivo com extensão `.world`, onde será inserido os artefatos do mundo, onde é descrito o robô, os itens que o povoam e o *layout* do mesmo.

O arquivo de extensão `.inc` segue a mesma sintaxe do `.world`, entretanto, neste pode ser incluído, por exemplo, se existe um objeto em um mundo e o usuário deseja utilizá-los em outros mundos (um robô, por exemplo), coloca-se a descrição específica do robô em um arquivo de extensão `.inc`, isso significa que, se o usuário desejar alterar a descrição do robô, somente será necessário alterar um único arquivo, de forma que suas múltiplas simulações são também alteradas.

Os arquivos de extensão `.cfg` contém todas as informações sobre o robô que será utilizado. Ele comunica ao *Player* quais *drivers* são necessários para interagir com o robô, se o usuário estiver usando um robô real, estes *drivers* são construídos no *Player*, alternativamente, se for uma simulação, o *driver* é sempre o *Stage* (isto é, o *Player* usa *Stage*, da mesma forma que usa um robô: ele pensa que é um *driver* de *hardware* e se comunica como tal). Em outras palavras, o arquivo `.cfg` informa ao *Player* como se comunicar com o *driver*, e como interpretar os dados a partir do *driver*, para que ele possa ser apresentado ao seu código. Itens descritos no arquivo `.world` devem ser descritos no arquivo `.cfg`, se o usuário desejar que seu código seja capaz de interagir com esse item (o robô).

C. Ambiente Integrado

Robôs móveis interagem com o ambiente através de sensores, que correspondem aos “sentidos” do robô, e atuadores, que permitem que o robô se movimente, manipule objetos ou direcione seus sensores para regiões de interesse. Dos sensores e atuadores mais comumente usados em robôs móveis, o *Player/Stage* simula de forma eficiente um determinado ambiente escolhido pelo desenvolvedor, ainda que não possa garantir seu comportamento de forma idêntica em um ambiente real, o uso dessa ferramenta auxilia no desenvolvimento de um projeto inicial.

Considerando essa capacidade de simulação, um robô

pode oferecer dispositivos para detectar sua posição, para definir sua altura, direção, identificar objetos e etc. Considerando assim o uso de sensores e atuadores tão funcionais quanto importantes. Podemos definir os sensores em dois tipos, sensores ativos, que são aqueles que emitem sinais de forma propositada para captar a informação de interesse. Tipicamente, estes sinais é despendido na forma de um pulso (luz, som, etc.), que é refletido por algum elemento do ambiente externo e é captada de volta pelo próprio sensor. O segundo tipo de sensor é o sensor passivo que pode ou não usar alguma forma de emissão de sinal para se manter operacional, ainda que nenhum sinal seja emitido pelo sensor com o objetivo explícito de obter uma reflexão detectável.

Entre os sensores ativos, os mais comuns são: sonar emissor infravermelho e laser. Entre os Sensores Passivos, os mais comuns são: odometria, sensor de choque, câmera e GPS (*Global Positioning System*).

Como mencionado anteriormente, os atuadores estão mais ligados às peças que envolvam movimento ou manipulação de objetos externos, no nosso caso, consideramos os atuadores de movimento como os mais importantes, sendo a roda, que é o mais usado em robôs, a configuração mais comum é a baseada em *drive* diferencial, na qual duas rodas são montadas em um eixo comum e comandadas por dois motores independentes. Sendo assim, o ambiente de simulação *Player/Stage* possui métodos que simulam os princípios citados acima, permitindo que usuário construa seu robô e/ou mundo da forma que preferir. Algumas funcionalidades:

- *Position2d*: dispositivo utilizado para se obter informações sobre a posição do robô (baseado no seu odômetro interno) e para mover o robô, calculando a distância do robô em relação a sua posição inicial;
- *Laser*: um sensor que mede a distância entre o robô e os obstáculos ao seu redor. O *laser* cobre um campo 180 graus, com 180 ou 360 leituras, fazendo leituras a cada 1 ou 0.5 graus;
- *Sonar*: semelhante ao *laser*, porém com um alcance menor e são apontados em direções específicas;
- *Blobfinder*: Utiliza imagens de uma câmera para localizar e identificar cores.

Atualmente o *Player* é compatível com diversas linguagens de programação tais como: C, C++, Java, Python, entre outras.

III. Ambiente de Desenvolvimento - *QTCreator*

A linguagem de programação escolhida para desenvolver o código que controla o robô foi $C++$. Para o desenvolvimento em $C++$ da aplicação proposta neste projeto utilizaremos o ambiente de desenvolvimento integrado ou IDE (*Integrated Development Environment*) *QTCreator*. O *QTCreator* é uma ferramenta especializada em desenvolvimento na plataforma QT que utiliza o $g++$(GCC), que é conjunto de compiladores compatíveis com a arquitetura dos principais processadores (CPU) do mercado e o GDB como depurador e interpretador de comandos, essa ferramenta oferece desenvolvimento de aplicações multi-plataforma de maneira eficiente e fácil, como sua principal característica, o alto desempenho. A versão da IDE utilizada neste trabalho é a *QTCreator* versão 4.5.

IV. O Modelo Proposto

Consideramos como etapa inicial deste modelo o processo de instalação e testes referente ao ambiente de simulação *Player/Stage* que, apesar de parecer simples, não é tão trivial. A partir da instalação foi possível a realização de estudos necessários à sua manipulação.

A etapa seguinte consiste em instalar a IDE *QTCreator*, utilizada para o desenvolvimento do código para controle do robô. É necessário também um estudo sobre a configuração da IDE, de forma a trabalhar em conjunto com a ferramenta de simulação.

Em seguida dar-se-á continuidade ao estudo das teorias necessárias para o desenvolvimento da pesquisa. Foi realizado um estudo especificamente sobre a teoria do funcionamento da ferramenta de simulação, a teoria da robótica, as linguagens de programação C e $C++$, além de partes específicas da documentação da IDE para uso das suas funcionalidades para desenvolvimento do aplicativo.

Realizados esses passos inicias, pode-se iniciar o processo de construção da simulação e o desenvolvimento do software para o controle do robô inserido no ambiente de simulação, com o objetivo de controlar o robô manualmente, por meio das teclas de navegação, no qual o robô é controlado por botões em um janela gráfica, conforme será apresentado a seguir.

Conforme proposto inicialmente, o resultado deste trabalho é o desenvolvimento de um software que permita controlar o robô manualmente. Dois pontos importantes devem ser destacados referentes ao desenvolvimento do aplicativo: o primeiro é a necessidade de manter dois processos funcionando paralelamente e, o segundo, a troca de mensagens entre eles, de forma que o funcionamento de cada processo dependa diretamente do funcionamento entre ambos.

Referente ao primeiro ponto, foi utilizado um conceito muito difundido na computação, o conceito de *thread*, uma forma de dividir o processo em vários fluxos de controle no mesmo espaço de endereçamento, executando quase em paralelo, como se fossem processos separados, permitindo que varias execuções ocorram no mesmo ambiente de processo de forma bastante independente umas da outras TANENBAUM (2000), como, por exemplo, a comunicação constante do *Player Client*. Nesse caso foi necessário a utilização da biblioteca `pthread.h`, desenvolvida para habilitar o uso de threads em softwares construídos nas linguagens $C/C++$ em sistemas operacionais Linux.

O segundo ponto foi resolvido com a criação de uma *struct* compartilhada entre o programa principal e o método

executado pela *thread*, conforme o seguinte código:

```
lstlisting
typedef struct{
double forwardSpeed, turnSpeed;
PlayerClient robot;
}ATRIBUTOS;
```

, onde a *struct* dever ser passada como parâmetro por referência para a *thread* no início da execução do programa, pois a *thread* é criada uma única vez no início da execução. Durante o processamento os campos da *struct* são alterados, como o método executado na *thread* é alimentado pela *struct*, o seu comportamento é alterado.

O comportamento descrito nos parágrafos anteriores é essencialmente o funcionamento do aplicativo e, para o seu funcionamento dois processos ocorrem em paralelo: a janela principal e o controle do robô. Na janela principal existem seis botões (Figura 2) com as seguintes funções:

- *Acelera*: faz o robô andar para frente;
- *Ré*: faz o robô andar para trás;
- *Direita*: faz o robô virar para direita;
- *Esquerda*: faz o robô virar para esquerda;
- *Para*: faz o robô parar;
- *Encerrar*: encerra o aplicativo.

Conforme apresentado na Figura 2, todos são botões de movimento, com exceção do botão *Encerrar*. Ao clicar em um botão de movimento, um valor é inserido em uma variável da *struct*, por exemplo, se o botão *Acelera* for clicado, a variável que armazena o valor da velocidade do robô recebe o valor 1 e então o robô terá velocidade 1.

Figura 2. Tela principal do modelo proposto

O segundo processo é responsável por aplicar na trajetória do robô as informações recebidas da tela principal e fazer funcionar a segunda tela, onde é apresentado a simulação, esta tela é iniciada pela ferramenta de simulação *Player/Stage* (Figura 3).

Este segundo processo possui, como mencionado anteriormente, a referência para a estrutura que é alterada no programa principal, além do que é executado independentemente na *thread*, criada pelo processo principal. O método executado na *thread* e, que controla o robô, é alimentado

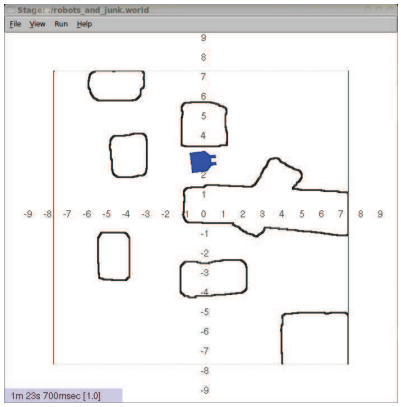

Figura 3. Tela de Simulação

todo o tempo pela *struct* da janela principal e em nenhum momento durante a execução do programa a *thread* é parada, sendo assim, qualquer alteração nos valores dos campos da *struct* afetará imediatamente o funcionamento do método executado pela *thread*, dessa forma alterando o movimento do robô. Podemos ver a seguir o código executado na *thread* que controla o robô.

```
void * caminha(void * args) {
ATRIBUTOS *at = (ATRIBUTOS *)args;
PlayerClient r;
Position2dProxy p(&at->robot,0);
p.SetMotorEnable(1);
while(true) {
at->robot.Read();
p.SetSpeed(at->forwardSpeed,
dtor(at->turnSpeed));
}
}
```

A struct Atributos contém o robô e as variáveis de avanço e giro do mesmo. Sendo assim, o método caminha recebe um ponteiro da struct Atributos, sendo responsável por acionar o motor do robô a partir do método SetMotorEnable(1), logo em seguida entra em um loop esperando por uma nova ação do usuário (método Read).

V. Conclusão

Apresentamos um aplicativo que apresenta uma interface simples, não apresentando dificuldades ao usuário, podendo ser utilizado para observação de trajetórias de robôs e servir de base para estudos mais aprofundados. Foram desenvolvidas as opções de acelerar, ré, rotacionar, parar, podendo ser manipulados diretamente através de eventos de entrada via cliques.

Concluímos como parte desse projeto de pesquisa, com um robô simulado no ambiente que pode ser controlado e direcionado manualmente por qualquer caminho para um

destino, evitando qualquer contato do robô. Além disso reunimos conhecimento a respeito de uma ferramenta de simulação importante para o desenvolvimento de pesquisas na área de robótica móvel, contribuindo assim consideravelmente para a comunidade acadêmica.

Dessa forma, podemos ter um alcance maior para pesquisas mais complexas na área de algoritmos, aplicando algoritmos voltados à robótica e evoluindo neste campo de pesquisa. Entre as prováveis pesquisas futuras estão: navegação autônoma de robôs e reconhecimento de ambiente utilizando métodos de aprendizagem.

REFERÊNCIAS

COLLETT, Toby H.J.; MACDONALD, B. A. and B. P. GERKEY (2005). Player 2.0: Toward a practical robot programming framework. *IEEE, In Proceedings of the Australasian Conference on Robotics and Automation (ACRA 2005), Sydney, Australia*.

OWEN, J. (2010). How to use player/stage 2nd edition. *The Player/Stage Manual*.

RIBEIRO, C., A. REALI, and R. ROMERO (2001). Robôs móveis inteligentes: Princípios e técnicas. *I Jornada de Atualização em Inteligência Artificial - JAIA2001, Anais do XXI Congresso da SBC 3*, 257–306.

TANENBAUM, Andrew S. e WOODHULL, A. S. (2000). *Sistemas operacionais : projeto e implementação*. 2ł ed. Porto Alegre: Bookman.

WOLF, Denis F.and OSÓRIO, F. S. S. E. and O. Trindade Jr. (2009). Robótica inteligente: Da simulação às aplicações no mundo real. in: André ponce de leon f. de carvalho; tomasz kowaltowski. (org.). *JAI: Jornada de Atualização em Informática da SBC. Rio de Janeiro: SBC - Editora da PUC Rio 1*, 279–330.